Técnicas de Busca e Ordenação – 2024/1

Trabalho Prático T3

19 de agosto de 2024

Leia atentamente **todo** esse documento de especificação. Certifique-se de que você entendeu tudo que está escrito aqui. Havendo dúvidas ou problemas, fale com o professor o quanto antes. As dúvidas podem ser sanadas usando o fórum de dúvidas do AVA.

1 Objetivo

O objetivo deste trabalho é aplicar o conceito de Tabela de Símbolos para o desenvolvimento de uma mini máquina de busca.

2 A anatomia de uma máquina de busca

Não vai parecer novidade para nenhum de vocês que desenvolver uma máquina de busca para a Web é uma tarefa complexa. E não apenas é uma tarefa complexa hoje, como também era quando as primeiras máquinas de busca surgiram. Para quem tiver o interesse, sugere-se a leitura do artigo escrito pelos criadores do Google no final da década de 90.[1]
Minimamente, uma máquina de busca deve ter os seguintes componentes:

1. *Web Crawler:* responsável por, constantemente, explorar a Web e obter o conteúdo das páginas;
2. Indexador: responsável por mapear os termos (ou combinações de termos) para as páginas que os contêm;
3. Ordenador de páginas: responsável por ordenar as páginas/documentos de acordo com usas respectivas importâncias. Esse componente é essencial para definir a ordem de apresentação das páginas que contêm um certo termo buscado.
4. Processador de consultas: dada uma consulta realizada por um usuário, este componente é responsável por identificar os documentos relacionados ao termo (de acordo com o indexador) e apresentá-los em ordem decrescente de importância (de acordo com o ordenador).

3 Tarefa de implementação

Neste trabalho, vocês deverão implementar versões simplificadas dos componentes 2, 3 e 4, descritos acima. Os requisitos das implementações são descritos nas Seções 3.1, 3.2 e 3.3.

3.1 Indexador

A entrada deste componente será uma coleção de documentos de texto $\{d_0, d_1, \ldots, d_{n-1}\}$, satisfazendo as propriedades abaixo.

- Cada documento será um arquivo de texto simples, ou seja, vocês não precisam se preocupar em fazer o *parser* de páginas HTML.
- Cada documento será uma sequência de termos separados por um, ou mais, caracteres delimitadores.

[1] `http://infolab.stanford.edu/~backrub/google.html`

- Os caracteres delimitadores válidos são o caractere de espaço, `\t` e `\n`.
- Cada termo será uma sequência dos seguintes tipos de caracteres: letras (minúsculas ou maiúsculas), números, e o hífen. Não haverá caracteres acentuados.

Além dos documentos, vocês receberão também uma lista de termos S, contendo *stop words*.[2]
Nesse componente, vocês deverão criar uma tabela de símbolos T que mapeie cada termo t, que apareça na coleção de documentos para a lista de documentos que contêm t. Além disso, todos os termos que estiverem em S deverão ser ignorados e não devem estar em T. A comparação de *strings* não deve distinguir letras maiúsculas e minúsculas (*case insensitive*).
Vocês serão livres para escolher a implementação de tabela de símbolos que quiserem, desde que seja uma das vistas durante o curso. Lembrem-se, no entanto, que uma implementação muito simplista pode não ser eficiente e **eficiência é um critério importante na nota**.

3.2 Google's PageRank

3.2.1 Intuição e definição

Quando um usuário digita um termo em uma máquina de busca, há interesse não apenas que os resultados obtidos sejam relevantes, mas também que os mais relevantes sejam apresentados primeiro. Em outras palavras, dadas as páginas que contêm o termo buscado (ou conteúdo relevante aos termos buscados), em que ordem essas páginas devem ser apresentadas pela máquina de busca?
Nos anos 90, havia um interesse muito grande em resolver esse problema, sendo que várias abordagens foram experimentadas. Por volta de 1997/1998, os criadores do Google propuseram o PageRank, método que serve como base para novos algoritmos até os dias de hoje.
O princípio básico do PageRank consiste no fato de que a importância de uma página Web não depende de seu conteúdo, autor, quem a possui, onde está hospedada, etc., mas sim de quão influente essa página é sobre outras páginas da Web. Nesse contexto, a influência de uma página sobre outra é dada pela existência de um *link* entre elas.
Por exemplo, suponha que a página principal do jornal *The New York Times* (`www.nytimes.com`) possua um *link* para a página principal da UFES (`www.ufes.br`), de forma que um visitante de `www.nytimes.com` possa clicar nesse *link* e possa então ir para `www.ufes.br`. Assim, é dito que `www.ufes.br` influencia `www.nytimes.com`, ou de forma recíproca, que `www.nytimes.com` passa parte de sua "autoridade" para `www.ufes.br`.

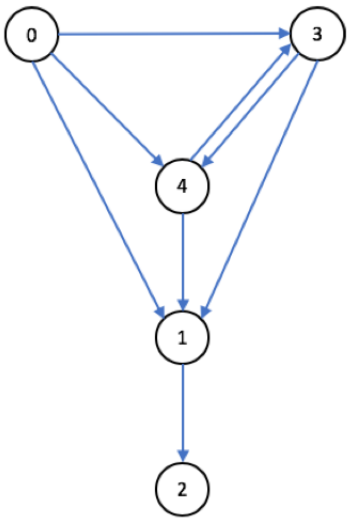

Figura 1: Exemplo de uma rede com 5 páginas.

[2] `https://en.wikipedia.org/wiki/Stop_word`

Dessa forma, as únicas coisas que o PageRank leva em consideração são um conjunto de páginas da Web (as quais se deseja ordenar) e a estrutura de *links* entre elas. A Figura 1 ilustra uma situação com 5 páginas. Na figura, uma seta de i para j indica que a página i contém um *link* para a página j. Dessa forma, a página 0 não influencia nenhuma outra página; A página 1 influencia as páginas 0, 3 e 4, sendo desta forma uma página importante na rede; A página 2 influencia apenas a página 1. Inicialmente, pode-se pensar que esta é uma página pouco relevante, no entanto, quando uma página i é influente sobre uma outra página influente j, tem-se que i também deve ser influente. Assim, a página 2 também é influente na rede do exemplo.
Para capturar a intuição apresentada no parágrafo anterior, o PageRank da página i é:

$$PR(i) = \begin{cases} \frac{1-\alpha}{n} + \alpha \sum\limits_{j \in In(i)} \frac{PR(j)}{|Out(j)|}, & \text{se } |Out(i)| \neq 0 \\ \frac{1-\alpha}{n} + \alpha PR(i) + \alpha \sum\limits_{j \in In(i)} \frac{PR(j)}{|Out(j)|}, & \text{se } |Out(i)| = 0. \end{cases} \tag{1}$$

Na definição acima:

- $PR(i)$ denota o PageRank da i-ésima página (ou seja, da página/documento d_i);
- n é o número total de páginas/documentos;
- α é um parâmetro, entre 0 e 1. Normalmente, α vale 0.85, sendo este o valor a ser usado neste trabalho;
- $In(k)$ é o conjunto de todas as páginas que têm um *link* para a página k;
- $Out(k)$ é o conjunto de todas as páginas com um *link* saindo da página k;
- $|Y|$ denota o número de elementos de um conjunto qualquer Y.

Aplicando a definição acima para a rede da Figura 1, tem-se que:

- $PR(0) = \frac{0.15}{5}$
- $PR(1) = \frac{0.15}{5} + 0.85 \left(\frac{PR(0)}{3} + \frac{PR(3)}{2} + \frac{PR(4)}{2} \right)$
- $PR(2) = \frac{0.15}{5} + +0.85PR(2) + 0.85 \left(\frac{PR(1)}{1} \right)$
- $PR(3) = \frac{0.15}{5} + 0.85 \left(\frac{PR(0)}{3} + \frac{PR(4)}{2} \right)$
- $PR(4) = \frac{0.15}{5} + 0.85 \left(\frac{PR(0)}{3} + \frac{PR(3)}{2} \right)$

Note que nesse caso, e na verdade no caso geral, a definição do valor do PageRank de cada página é cíclica. Ou seja, não há uma fórmula fechada para calculá-lo.

3.2.2 Como calcular o PageRank?

Para calcular o PageRank, vocês deverão implementar um método iterativo conhecido como *Power Method*. O algoritmo tem um passo de inicialização dos valores do PageRank e então entra em um laço para atualizar tais valores até que convergência seja obtida. Em detalhes:

- Inicialização. No passo zero, o PageRank de cada página é inicializado de maneira uniforme:

$$PR^{(0)}(i) = \frac{1}{n}, i = 0, \ldots, n-1. \tag{2}$$

- Laço de atualização. Na iteração $k \geq 1$, o valor do PageRank é computado com base nos valores da iteração anterior:

$$PR^{(k)}(i) = \begin{cases} \frac{1-\alpha}{n} + \alpha \sum\limits_{j \in In(i)} \frac{PR^{(k-1)}(j)}{|Out(j)|}, & \text{se } |Out(i)| \neq 0 \\ \frac{1-\alpha}{n} + \alpha PR^{(k-1)}(i) + \alpha \sum\limits_{j \in In(i)} \frac{PR^{(k-1)}(j)}{|Out(j)|}, & \text{se } |Out(i)| = 0, \end{cases} \tag{3}$$

para $i = 0, \ldots, n-1$. Repare que essa equação é similar à definição do PageRank. Mas agora, os valores de PageRank da iteração $k-1$ são utilizados para computar os valores do PageRank da iteração k.

- Término. O processo iterativo de atualização deve ser encerrado quando os valores do PageRank pararem de mudar significativamente. É possível mostrar que isso sempre ocorrerá. De forma prática, o critério de parada será baseado na quantidade a seguir:

$$E(k) = \frac{1}{n} \sum_{i=0}^{n-1} |PR^{(k)}(i) - PR^{(k-1)}(i)|. \quad (4)$$

Dado $E(k)$, o algoritmo para se $E(k) < \epsilon$, onde ϵ é uma constante pequena, e.g., 10^{-6}. Quando tal fato ocorrer, o valor de $PR^{(k)}(i)$ será tomado como o valor do PageRank da página i, $i = 0, \ldots, n-1$.

Considerando os valores de parâmetros descritos acima e o exemplo usando nas seções anteriores (Figura 1), tem-se o exemplo de computação dado na Tabela 1:

Tabela 1: Cálculo do PageRank para a rede da Figura 1 (valores arredondados por questão de espaço).

k	$PR^{(k)}$	$E(k)$
0	[0.2 0.2 0.2 0.2 0.2]	-
1	[0.03, 0.25666667, 0.37, 0.17166667, 0.17166667]	0.09066667
2	[0.03, 0.18441667, 0.56266667, 0.111458333, 0.11145833]	0.07706667
...	...	...
16	[0.03, 0.09541360, 0.74067280, 0.06695680, 0.06695680]	6.04e-07

3.3 Processador de consultas

A entrada deste componente será uma sequência de termos $t_0, t_1, \ldots t_{m-1}$, os quais representam uma frase ou conjunto de termos sendo buscados na máquina de busca. É garantido que cada consulta terá ao menos um termo que não seja uma *stop word*.
A saída deste componente (e do trabalho em si) será um subconjunto R, dos documentos $d_0, \ldots, d_{n-1}$, que satisfaça:

1. Cada documento em R deve conter todos os termos da busca (que não sejam *stop words* – i.e., *stop words* devem ser ignoradas). Não é necessário que os termos apareçam, em um documento, juntos ou na mesma ordem da busca. Além disso, a busca é *case insensitive*.

2. Os documentos devem ser listados em ordem decrescente de PageRank. Empates no PageRank devem ser quebrados pela ordem lexicográfica, crescente, do nome dos documentos.

4 Exemplo

Como exemplo, considere um conjunto de 5 páginas/documentos {`a.txt`, `b.txt`, `c.txt`, `d.txt`, `e.txt`}. O conteúdo de cada página é listado abaixo:

- `a.txt`: “O abacate e uma fruta boa”;
- `b.txt`: “O abacate e uma fruta ruim”;
- `c.txt`: “Eu gosto de Abacate abacaxi e ruim”;
- `d.txt`: “eu odeio abacate”;
- `e.txt`: “Maca e melhor que abacate”;

Considere também que $S =$ {“de”, “que”, “e”, “o”, “uma”, “eu”}. Assim, a tabela T conterá o seguinte mapeamento:

- "abacate" → {`a.txt`, `b.txt`, `c.txt`, `d.txt`, `e.txt`};
- "fruta" → {`a.txt`, `b.txt`};
- "boa" → {`a.txt`};
- "ruim" → {`b.txt`, `c.txt`};
- "gosto" → {`c.txt`};
- "abacaxi" → {`c.txt`};
- "odeio" → {`d.txt`};
- "maca" → {`e.txt`};
- "melhor" → {`e.txt`};

Considere ainda que estas 5 páginas tenham uma estrutura de *links* similar à estrutura da Figura 1. Ou seja:

- `a.txt` tem *links* para {`b.txt`, `d.txt`, `e.txt`};
- `b.txt` tem *links* para {`c.txt`};
- `c.txt` tem *links* para {};
- `d.txt` tem *links* para {`b.txt`, `e.txt`};
- `e.txt` tem *links* para {`b.txt`, `d.txt`};

Assim, se a busca "maca abacate" for realizada, o resultado deve ser [`e.txt`]. Se a busca "abacate ruim" for realizada, o resultado deve ser [`c.txt`, `b.txt`]. E se a busca pelo termo "liquidificador" for realizada, o resultado deve ser [].

5 Entrada e Saída

Leia essa seção com atenção. Não imprima nada na tela e não solicite nenhuma informação que não esteja aqui descrita.

5.1 Entrada

Você deve criar um programa que leia, da linha de comando, um argumento representando o nome de um diretório. Esse diretório deverá ter o conteúdo listado abaixo.

- Um arquivo chamado `index.txt`. Esse arquivo possui o nome, um por linha, de todas as páginas/documentos a serem considerados.
- Um arquivo chamado `stopwords.txt`. Esse arquivo possui todas as *stop words*, uma por linha.
- Um arquivo chamado `graph.txt`. Esse arquivo possui a estrutura de *links* entre as páginas. Cada linha desse arquivo possui: o nome da página de origem, o número de *links* de saída dessa página e a lista de páginas para a qual ela aponta.
- Um diretório chamado `pages`. Esse diretório contém todos os arquivos de texto listados em `index.txt`.

Após iniciar, seu programa deve ler uma sequência de consultas, uma por linha, da entrada padrão. Cada consulta será uma lista de termos separados por caracteres de espaço. O programa deve continuar lendo consultas até que o caractere de fim de arquivo seja lido.
Por exemplo, no caso da seção anterior, tem-se:

- Conteúdo do arquivo `index.txt`

```
a.txt
b.txt
d.txt
e.txt
c.txt
```

- Conteúdo do arquivo `stopwords.txt`

```
de
que
e
o
uma
eu
```

- Conteúdo do arquivo `graph.txt`

```
a.txt 3 b.txt d.txt e.txt
b.txt 1 c.txt
c.txt 0
d.txt 2 b.txt e.txt
e.txt 2 b.txt d.txt
```

- Conteúdo do diretório `pages`

```
a.txt b.txt c.txt d.txt e.txt
```

- Conteúdo do arquivo `pages/a.txt`

```
O abacate e uma fruta boa
```

- Conteúdo do arquivo `pages/b.txt`

```
O abacate e uma fruta ruim
```

- Conteúdo do arquivo `pages/c.txt`

```
Eu gosto de Abacate abacaxi e ruim
```

- Conteúdo do arquivo `pages/d.txt`

```
eu odeio abacate
```

- Conteúdo do arquivo `pages/e.txt`

```
Maca e melhor que abacate
```

5.2 Saída

A saída deve ser feita na saída padrão. Para cada consulta, o resultado deve ser composto de três linhas. A primeira linha identifica a consulta. A segunda linha é lista de arquivos (que contêm os termos buscados) impressos (em uma única linha) separados por um único caractere de espaço e ordenados em ordem decrescente de PageRank. A terceira linha deve conter os PageRanks das respectivas páginas listadas na linha anterior (na mesma ordem). Em caso de empate no PageRank, deve-se seguir a ordem lexicográfica crescente do nome dos documentos. Abaixo, a saída para o caso dos exemplos anteriores. Você deve seguir exatamente esse mesmo formato.

- A consulta “maca abacate” tem resultado

```
search:maca abacate
pages:e.txt
pr:0.06695664
```

- A consulta “abacate ruim” tem resultado

```
search:abacate ruim
pages:c.txt b.txt
pr:0.74067344 0.09541328
```

- A consulta “liquidificador” tem resultado

```
search:liquidificador
pages:
pr:
```

Para testar seu trabalho, o professor executará comandos seguindo o seguinte padrão.

```
tar -xzvf <nome_arquivo>.tar.gz
make
./trab3 <diretório_entrada>
```

É extremamente importante que vocês sigam esse padrão. Seu programa não deve solicitar entradas ou imprimir nada que não seja exigido nesta especificação.
Por exemplo, se o nome do arquivo recebido for `2004209608.tar.gz` e os dados de entrada estiverem no diretório `/tmp/data/`, o professor vai digitar:

```
tar -xzvf 2004209608.tar.gz
make
./trab3 /tmp/data/
```

6 Detalhes de implementação

A seguir, alguns detalhes, comentários e dicas sobre a implementação. Muita atenção aos usuários do Sistema Operacional Windows.

- o trabalho deve ser implementado em C. A versão do C a ser considerada é a presente no Sistema Operacional Ubuntu instalado nos computadores do LabGrad1.
- o caractere de nova linha será o `\n`.
- Seu programa deve ser, obrigatoriamente, compilado com o utilitário `make`. Crie um arquivo `Makefile` que gera como executável para o seu programa um arquivo de nome `trab3`.
- Ao longo do desenvolvimento do trabalho, certifique-se que o seu código não está vazando memória testando-o com o `valgrind`. Não espere terminar o código para usar o `valgrind`, incorpore-o no seu ciclo de desenvolvimento. Ele é uma ferramenta excelente para se detectar erros sutis de acesso à memória que são muito comuns em C. Idealmente o seu programa deve sempre executar sem nenhum erro no `valgrind`.
- Não é necessária nenhuma estrutura de dados muito elaborada para o desenvolvimento deste trabalho. Todas as estruturas que você vai precisar foram discutidas em aula ou no laboratório. Veja os códigos disponibilizados pelo professor para ter ideias. Prefira estruturas simples a coisas muito complexas. Pense bem sobre as suas estruturas e algoritmos antes de implementá-los: quanto mais tempo projetando adequadamente, menos tempo depurando o código depois.

7 Regras para desenvolvimento e entrega do trabalho

- **Data da Entrega**: O trabalho deve ser entregue até as 23:59h do dia 19/09/2024. Não serão aceitos trabalhos após essa data.
- **Grupo**: O trabalho pode ser feito em grupos de até quatro pessoas. **Não há restrição para formação de grupos nesse trabalho!**
- **Como entregar**: Pela atividade criada no AVA. Envie um arquivo compactado, no formato `.tar.gz`, com todo o seu trabalho. A sua submissão deve incluir todos os arquivos de código e um `Makefile`, como especificado anteriormente. **Somente uma pessoa do grupo deve enviar o trabalho no AVA. Coloque a matrícula de todos integrantes do grupo (separadas por vírgula) no nome do arquivo do trabalho.**
- **Recomendações**: Modularize o seu código adequadamente. Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Não deixe para começar o trabalho na última hora.

8 Relatório de resultados

Esse trabalho não demandará relatório.

9 Avaliação

- Assim como especificado no plano de ensino, o trabalho vale 10 pontos.
- A parte de implementação será avaliada de acordo com a fração e tipos de casos de teste que seu trabalho for capaz de resolver de forma correta. Casos *pequenos* e *médios* (4 pontos) serão utilizados para aferir se seu trabalho está correto. Casos *grandes* (4 pontos) serão utilizados para testar a eficiência do seu trabalho. Casos *muito grandes* (2 pontos) serão utilizados para testar se seu trabalho foi desenvolvido com muito cuidado e tendo eficiência máxima como objetivo. Todos os casos de teste serão projetados para serem executados em poucos minutos (no máximo 5) em uma máquina com 16GB de RAM.
- Trabalhos com erros de compilação receberão nota zero.
- Trabalhos que gerem *segmentation fault* para algum dos casos de teste disponibilizados no AVA serão severamente penalizados na nota.
- Trabalhos com *memory leak* (vazamento de memória) sofrerão desconto na nota.
- O uso de variáveis globais ou da primitiva `goto` implica em nota zero.
- Organização do código e comentários valem nota. Trabalhos confusos e sem explicação sofrerão desconto na nota.
- Caso seja detectada **cópia** (entre alunos ou da Internet), todos os envolvidos receberão nota zero. Caso as pessoas envolvidas em suspeita de cópia discordem da nota, amplo direito de argumentação e defesa será concedido. Neste caso, as regras estabelecidas nas resoluções da UFES serão seguidas.
- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre cópia, acima.)